Building a Brain in 10 Minutes

Many decades ago, artificial neural networks were developed to mimic the learning capabilities of humans and animals. Below is an excerpt from The Machine that Changed the World, a 1992 documentary about Artificial Intelligence.

```
from IPython.display import YouTubeVideo
YouTubeVideo('cNxadbrN_aI')
```

Since then, computers and machine learning libraries have evolved to where we can replicate many days of experimentation in just a few minutes. In this notebook, we will step through how artificial neural networks have improved over the years and the biological inspiration behind it.

To demonstrate, we will be using TensorFlow, an open-source machine learning library popular in industry. Recent versions of TensorFlow automatically detect if there is a GPU available for computation.

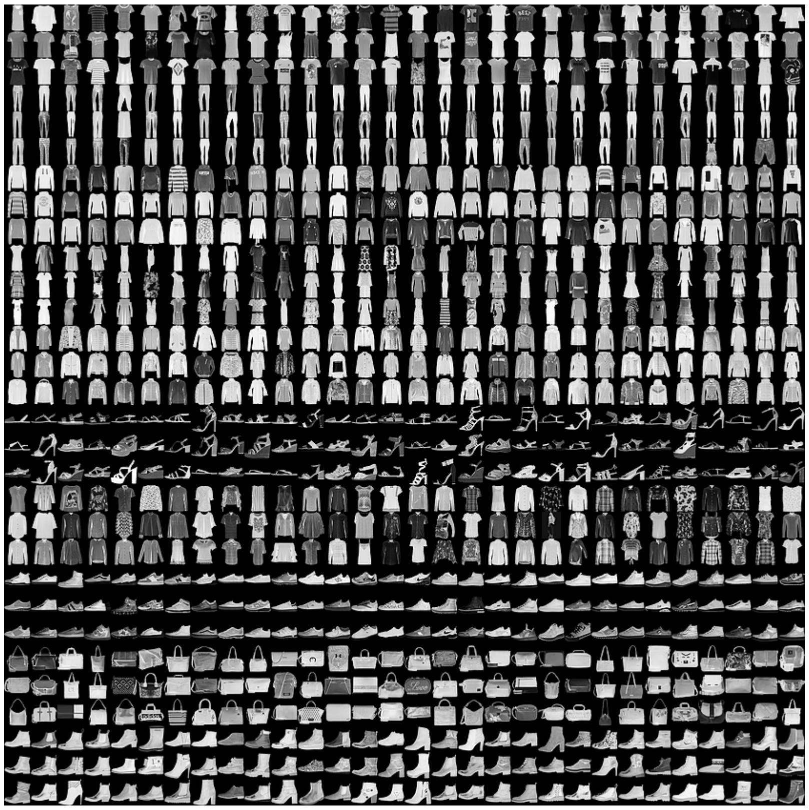
```
import tensorflow as tf

tf.config.list_physical_devices('GPU')
```

```
    [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

GPUs were originally designed for the significant amount of matrix mathematics used when rendering computer graphics. Neural networks also require a significant amount of matrix multiplication, making GPUs a good fit when building them.

Data

Speaking of graphics, we're going to tackle a challenge that seemed almost impossible decades ago: image classification with computer vision. Specifically, we will try to classify articles of clothing from the Fashion MNIST dataset. A few samples are shown below:



Neural networks attempt to copy the human learning technique, Trial and Error. To do this, we will create something like a set of digital flashcards. Our artificial brains will attempt to guess what kind of clothing we are showing it with a flashcard, then we will give it the answer, helping the computer learn from its successes and mistakes.

Just like how students are quizzed to test their understanding, we will set aside a portion of our data to quiz our neural networks to make sure they understand the concepts we're trying to teach them, as opposed to them memorizing the answers to their study questions. For trivia, memorization might be an acceptable strategy, but for skills, like adding two numbers, memorization won't get our models very far.

The study data is often called the `training dataset` and the quiz data is often called the `validation dataset`. As Fashion MNIST is a popular dataset, it is already included with the TensorFlow library. Let's load it into our coding environment and take a look at it.

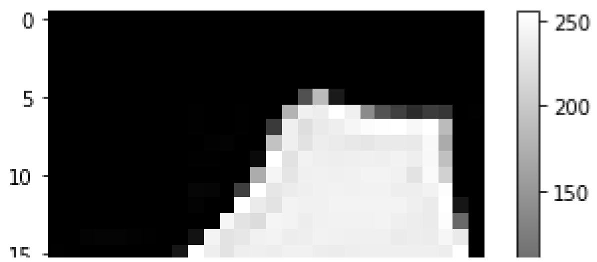
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (valid_images, valid_labels) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
32768/29515 [=================================] - 0s 0us/step
40960/29515 [=========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
26427392/26421880 [==============================] - 1s 0us/step
26435584/26421880 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-l
16384/5148 [==============================================================================
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-i
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

Let's start with our `train_images` and `train_labels`. `train_images` are like the question on our flashcards and `train_labels` are like the answer. In general, data scientists often refer to this answer as the `label`.

We can plot one of these images to see what it looks like. To do so, we will use Matplotlib.

```
import matplotlib.pyplot as plt

# The question number to study with. Feel free to change up to 59999.
data_idx = 42

plt.figure()
plt.imshow(train_images[data_idx], cmap='gray')
plt.colorbar()
plt.grid(False)
plt.show()
```



What would you classify this as? To make it easier, here are the ten categories it could be:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Have an answer? Run the following code cell to see if you were correct:

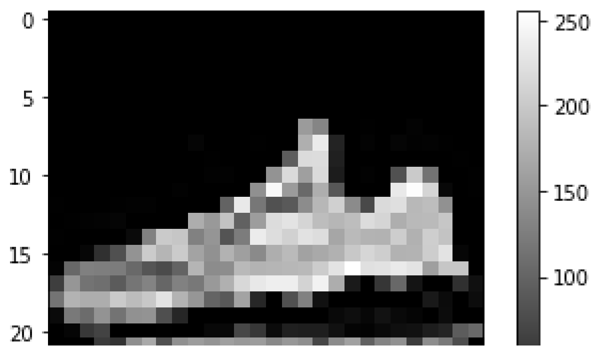
```
train_labels[data_idx]
```

```
     9
```

How did you do? Good job if you got it correct!

Our validation data has a similar structure of questions and answers:

```
import matplotlib.pyplot as plt

# The question number to quiz with. Feel free to change up to 9999.
data_idx = 6174

plt.figure()
plt.imshow(valid_images[data_idx], cmap='gray')
plt.colorbar()
plt.grid(False)
plt.show()
```



```
valid_labels[data_idx]
```

```
    7
```

Building a Neuron

Neurons are the fundamental building blocks to a neural network. Just like how biological neurons send an electrical impulse under specific stimuli, artificial neural networks similarly result in a numerical output with a given numerical input.

We can break down building a neuron into 3 steps:

- Defining the architecture
- Intiating training
- Evaluating the model

Defining the architecture

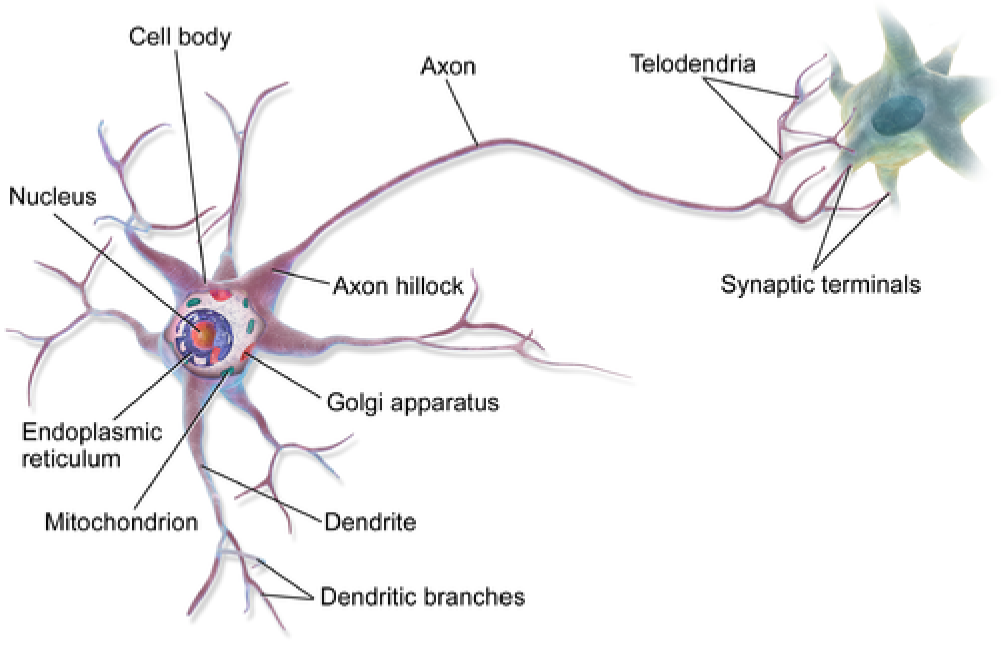


Image courtesy of [Wikimedia Commons]

Biological neurons transmit information with a mechanism similar to Morse Code. It receives electrical signals through the dendrites, and under the right conditions, sends an electrical impulse down the axon and out through the terminals.

It is theorized the sequence and timing of these impulses play a large part of how information travels through the brain. Most artificial neural networks have yet to capture this timing aspect of biological neurons, and instead emulate the phenomenon with simpler mathematical formulas.

The Math

Computers are built with discrete 0s and 1s whereas humans and animals are built on more continuous building blocks. Because of this, some of the first neurons attempted to mimic biological neurons with a linear regression function: `y = mx + b`. The `x` is like information coming in through the dendrites and the `y` is like the output through the terminals. As the computer guesses more and more answers to the questions we present it, it will update its variables (`m` and `b`) to better fit the line to the data it has seen.

Neurons are often exposed to multivariate data. We're going to build a neuron that takes each pixel value (which is between `0` and `255`), and assign it a weight, which is equivalent to our `m`. Data scientists often express this weight as `w`. For example, the first pixel will have a weight of `w0`, the second will have a weight of `w1`, and so on. Our full equation becomes `y = w0x0 + w1x1 + w2x2 + ... + b`.

Each image is 28 pixels by 28 pixels, so we will have a total of 784 weights. A pixel value of `0` would be black and a pixel value of `255` would be white. Let's look at the raw pixel values of the previous image we plotted. Each number below will be assigned a weight.

```
# 28 lists with 28 values each
valid_images[data_idx]
```

```
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,
          0,   0,  93, 219, 220,  32,   0,   0,   0,   0,   0,   0,   0,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   2,   2,   0,
          0, 147, 218, 159, 218,  28,   0,   2,   0,  82, 202, 115,   0,
          0,   1],
       [  0,   0,   0,   0,   0,   0,   0,   0,   2,   0,   0,   0,   0,
        144, 248, 155, 105, 174,  87,   0,   0,   0, 233, 253, 213,   8,
          0,   1],
       [  1,   0,   0,   0,   1,   2,   2,   0,   0,   0,   0,  99, 233,
        119,  81,  89, 140, 183, 208, 139,  74, 218, 223, 184, 189, 145,
          0,   0],
       [  0,   1,   2,   3,   5,   0,   0,   0,   0, 173, 153, 195,  97,
        157, 220, 228, 213, 189, 182, 189, 220, 213, 174, 185, 186, 198,
          0,   0],
       [  1,   0,   0,   0,   0,  10, 129, 202, 198, 129, 149,  84, 124,
        236, 221, 209, 226, 221, 166, 186, 179, 178, 205, 177, 206, 196,
          0,   0],
       [  0,   0,   6,  49,  80, 148, 203, 181, 203, 164, 145, 178, 159,
        138, 174, 186, 162, 169, 177, 200, 214, 206, 178, 179, 205, 227,
          6,   0],
       [  3, 104, 128, 126, 124, 105,  85,  75,  99, 181, 140, 141, 185,
        179, 181, 183, 185, 204, 227, 255, 224, 226, 234, 227, 159, 198,
        198,   0],
       [ 62, 150, 115, 111,  91, 117, 131, 102, 129, 120, 153, 176, 216,
        232, 238, 238, 211, 242, 173,  26,   0,  54, 104,  20,   0,   0,
         48,  16],
       [116, 179, 173, 173, 202, 188, 200, 228, 179, 150, 100,  89, 164,
         39,   0,  82, 129,  17,   0,   0,   0,   0,   0,   0,  25,  11,
          0,  12],
       [ 10, 122, 108, 148, 115, 145, 148,  79,  38,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   9,  14,   7,  18,   9,   5,   0,  12,
          0,   0],
       [  0,   5,  57,  67,   0,   0,   0,   1,   0,   0,   8,   8,  71,
         55,  11,  32,  30,  30,  17,   4,   3,   9,  15,  17,  29,  37,
         92, 106],
       [  0,   0,   0,  58,  50, 142, 169,  80, 144, 156, 148, 159, 152,
        156, 136, 151, 142,  70, 139, 161,  98, 131, 145,  60, 141, 121,
         77,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],

       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

One more thing to think about: the output of `y = mx + b` is a number, but here, we're trying to classify different articles of clothing. How might we convert numbers into categories?

Here is a simple approach: we can make ten neurons, one for each article of clothing. If the neuron assigned to "Trousers" (label #1), has the highest output compared to the other neurons, the model will guess "Trousers" for the given input image.

Keras, a deep learning framework that has been integrated into TensorFlow, makes such a model easy to build. We will use the Sequential API, which allows us to stack layers, the list of operations we will be applying to our data as it is fed through the network.

In the below model, we have two layers:

- Flatten - Converts multidimensional data into 1 dimensional data (ex: a list of lists into a single list).
- Dense - A "row" of neurons. Each neuron has a weight (`w`) for each input. In the example below, we use the number `10` to place ten neurons.

We will also define an `input_shape` which is the dimensions of our data. In this case, our `28x28` pixels for each image.

```
number_of_classes = train_labels.max() + 1
number_of_classes
```

```
10
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(number_of_classes)
])
```

Verifying the model

To make sure our model has the structure we expect, we can call the summary method.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense (Dense)               (None, 10)                7850      
                                                                 
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

We can see that our total parameter count is `7850`. Let's see if this makes sense. For each pixel, there should be a weight for each of our ten classes.

```
image_height = 28
image_width = 28
```

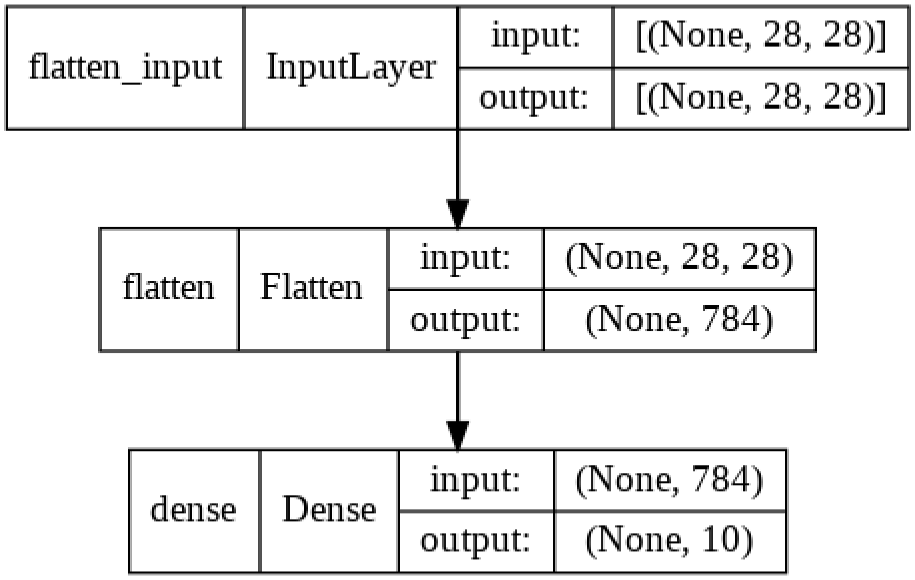
```
number_of_weights = image_height * image_width * number_of_classes
number_of_weights
```

```
    7840
```

So our weights make up `7,840` parameters. Where do the other ten come from? It's each of the `10` neurons biases, the `b` in `y = mx + b`.

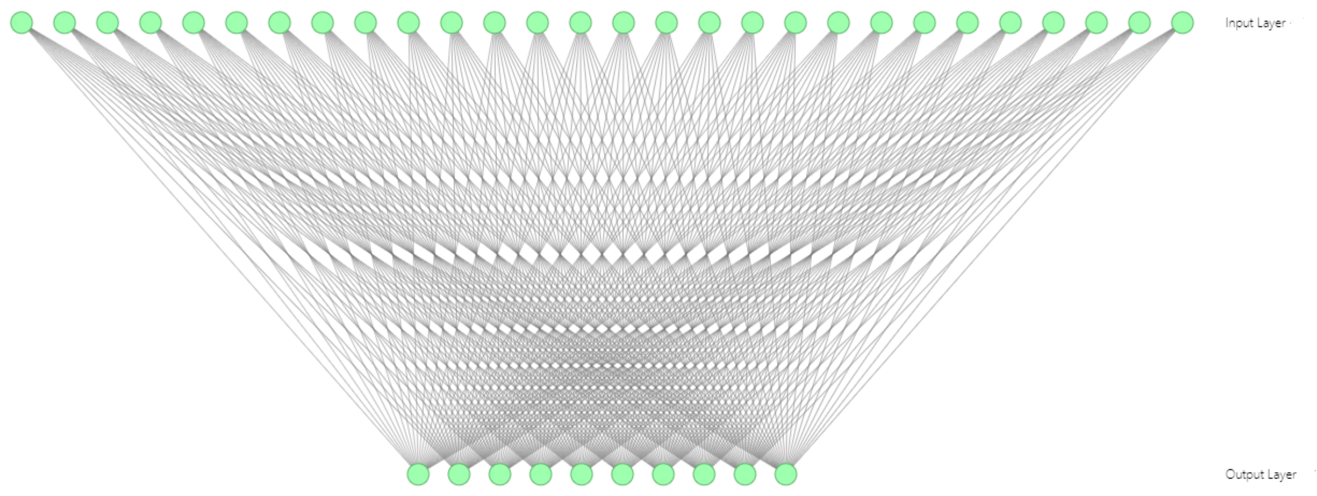
There are a few other ways to verify our model. We can also plot it:

```
tf.keras.utils.plot_model(model, show_shapes=True)
```



In academic papers, models are often represented like the picture below. In practice, modern neural networks are so large, it's impractical to graph them in this way. The below is a fraction of our entire model. There are 10 neurons on the bottom representing each of our ten classes, and 28 input nodes on the top, representing a row of our pixels. In reality, the top layer is 28 times bigger!

Each circle represents a neuron or an input, and each line represents a weight. The below image was created with a tool by Alex Lenail.



Initiate Training

We have a model setup, but how does it learn? Just like how students are scored when they take a test, we need to give the model a function to grade its performance. Such a function is called the `loss` function.

In this case, we're going to use a type of function specific to classification called SparseCategoricalCrossentropy:

- **Sparse** - for this function, it refers to how our label is an interger index for our categories
- **Categorical** - this function was made for classification
- **Cross-entropy** - the more confident our model is when it makes an incorrect guess, the worse its score will be. If a model is 100% confident when it is wrong, it will have a score of negative infinity!
- `from_logits` - the linear output will be transformed into a probability which can be interpreted as the model's confidence a particular category is the correct one for the given input.

This type of loss function works well for our case as it will grade each of the neurons simultaneously. If all of our neurons give a strong signal that they're the correct label, we need a way to tell them that they can't all be right.

For us humans, we can add additional `metrics` to monitor how well our model is learning. For instance, maybe the loss is low, but what if the `accuracy` is not high?

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Evaluating the model

Now the moment of truth! The below fit method will both help our model study and quiz it.

An `epoch` is one review of the training dataset. Just like how school students might need to review a flashcard multiple times before the concept "clicks", the same is true of our models.

After each `epoch`, the model will be quizzed with the validation data. Let's watch it work hard and improve:

```
history = model.fit(
    train_images,
    train_labels,
    epochs=5,
    verbose=True,
    validation_data=(valid_images, valid_labels)
)
```

```
Epoch 1/5
1875/1875 [==============================] - 8s 3ms/step - loss: 16.6021 - accuracy: 0.7
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 12.3148 - accuracy: 0.7
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 11.0784 - accuracy: 0.7
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 10.9143 - accuracy: 0.8
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 10.3428 - accuracy: 0.8
```

How did the model do? B-? To give it credit, it only had `10` neurons to work with. Us humans have billions!

The accuracy should be around 80%, although there is some random variation based on how the flashcards are shuffled and the random value of the weights that were initiated.

Prediction

Time to graduate our model and let it enter the real world. We can use the predict method to see the output of our model on a set of images, regardless of if they were in the original datasets or not.

Please note, Keras expects a batch, or multiple datapoints, when making a prediction. To make a prediction on a single point of data, it should be converted to a batch of one datapoint.

Below are the predictions for the first ten items in our training dataset.

```
model.predict(train_images[0:10])
```

```
     array([[ -253.19922  ,  -496.62082  ,   -52.590397 ,  -210.747    ,
              -204.85382  ,   434.7374   ,   119.23126  ,   455.66464  ,
               257.7323   ,   646.0232   ],
            [  276.18967  ,  -384.84845  ,   201.0406   ,    -3.3794174,
               -55.13553  , -1426.099    ,   209.81992  , -1563.9044   ,
               -99.46313  ,  -742.6002   ],
            [   64.293144 ,    -4.590648 ,    79.44332  ,   101.15966  ,
                88.32608  ,  -549.18677  ,    74.816605 ,  -503.69385  ,
               -70.380905 ,  -239.93843  ],
            [  116.54942  ,   -42.0613   ,   136.45375  ,   160.62233  ,
                91.001656 ,  -861.4117   ,   158.15169  ,  -894.7206   ,
              -135.96779  ,  -682.5919   ],
            [  202.69241  ,   218.59782  ,   184.85669  ,   268.64322  ,
               263.1653   , -1028.9443   ,   174.21268  ,  -702.1034   ,
                40.39121  ,  -782.7997   ],
            [   19.20883  ,  -182.54137  ,   306.686    ,    13.147002 ,
               181.66083  ,  -964.8558   ,   250.7611   , -1359.0271   ,
               -28.114925 , -1040.5476   ],
            [ -153.14479  ,  -231.27252  ,   -48.053577 ,   -96.47466  ,
               -80.90507  ,   264.1215   ,     2.1044562,   316.46457  ,
               162.20761  ,   179.98196  ],
            [ -120.6378   ,  -332.05634  ,   307.72177  ,  -234.34274  ,
               357.56802  , -1302.7218   ,   324.34933  , -1741.5159   ,
                14.074064 ,  -766.1739   ],
            [  -19.258186 ,  -351.6337   ,  -115.03816  ,  -145.74231  ,
              -184.4867   ,   465.18402  ,   -31.01576  ,    -5.0114846,
                29.787205 ,   235.35136  ],
            [ -180.77887  ,  -249.59503  ,    78.87244  ,   -87.607254 ,
              -105.95488  ,   904.1569   ,   150.95465  ,   -79.53638  ,
               215.98877  ,   322.95093  ]], dtype=float32)
```

These are raw results and need some formatting to be interpreted by the average human, so below, we've displayed an image to be classified as well as graph the results of each of our output neurons. The larger the value, the more confident the neuron is that it corresponds to the correct label (and the more negative it is, the more confident it is that it is **not** the correct label).

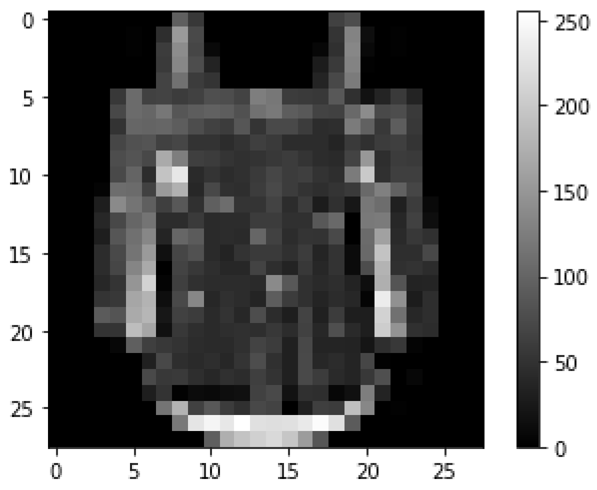
Alter the `data_idx` below to see how the model predicts against various images. How do you think it did? For the predictions it got wrong, would you have been confused too?

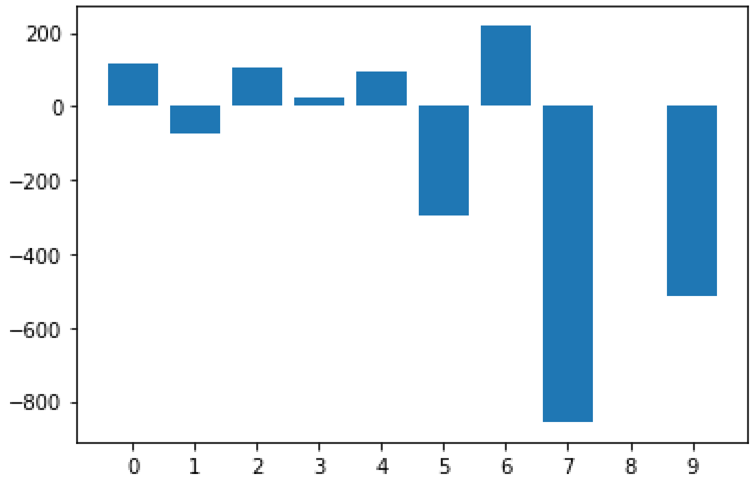
Here's the table again for reference:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt

Label	Description
7	Sneaker
8	Bag
9	Ankle boot

```
data_idx = 8675 # The question number to study with. Feel free to change up to 59999.

plt.figure()
plt.imshow(train_images[data_idx], cmap='gray')
plt.colorbar()
plt.grid(False)
plt.show()

x_values = range(number_of_classes)
plt.figure()
plt.bar(x_values, model.predict(train_images[data_idx:data_idx+1]).flatten())
plt.xticks(range(10))
plt.show()

print("correct answer:", train_labels[data_idx])
```





```
correct answer: 6
```

Conclusion

Congratulations on completing the notebook! While this model does significantly better than random guessing, it has a way to go before it can beat humans at recognizing clothing. Want to make computer vision models that can surpass humans in image classification? Learn more about the mechanics of neural networks and how to make more accurate models in our self-paced [Getting Started with Deep Learning](#) online course.

NVIDIA. | DEEP LEARNING INSTITUTE